Priority Search Trees

1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

MCS 481 Lecture 31
Computational Geometry
Jan Verschelde, 2 April 2025

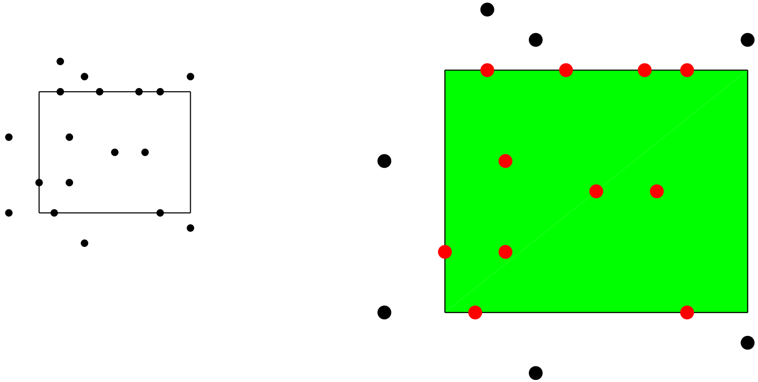
Priority Search Trees

1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

windowing queries

Given a map, we zoom in on a window $\Rightarrow$ windowing query.



We focus on points, although the points are end points of segments.

Motivation: reduce the storage from $O(n \log(n))$ to $O(n)$.

problem statement

Input: $P = \{ p_1, p_2, \ldots, p_n \}$, a set of n points in the plane; and
a query window $W = (-\infty : q_x] \times [q_y : q'_y]$.
Output: $P \cap W$.

Using a 2D range tree requires $O(n \log(n))$ storage because of the associated binary search trees.

How to integrate the information about y-coordinates into one structure, *without* associated structures?

Motivation: reduce the storage from $O(n \log(n))$ to $O(n)$.

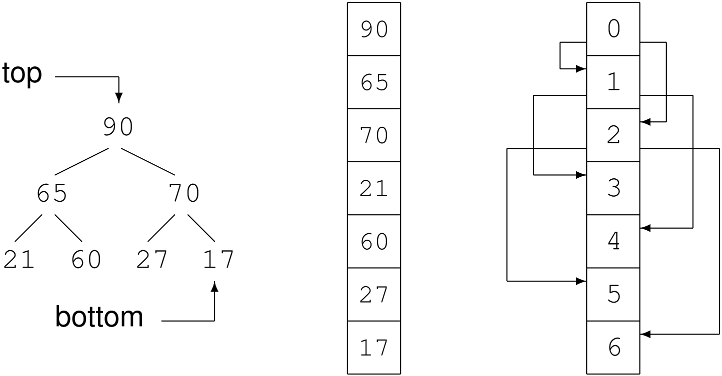
Priority Search Trees

1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

the heap

The *heap* or priority queue is a binary tree
where every node has a higher priority than its children.

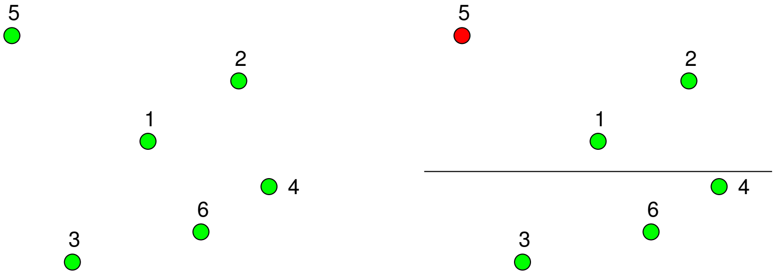


For node at p: left child is at $2p+1$, right child is at $2p+2$.
Parent of node at p is at $(p-1)/2$. Storage cost is $O(n)$.

the construction of a heap to store points

Construct a heap to store points as follows:

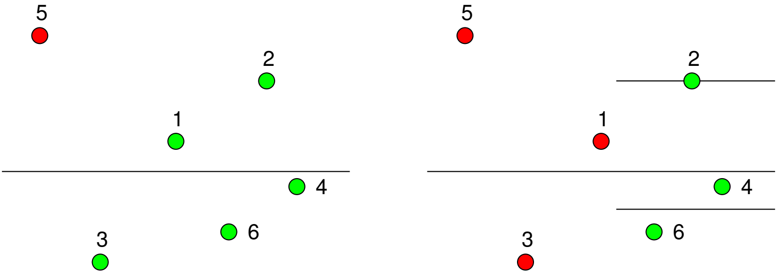
- take the leftmost point,
- split the other points on their median y-coordinate, and
- continue the construction recursively on the splitted halves.



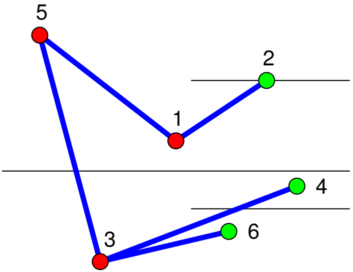
the construction of a heap continued

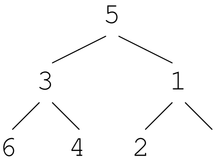
Construct a heap to store points as follows:

- take the leftmost point,
- split the other points on their median y-coordinate, and
- continue the construction recursively on the splitted halves.



a heap for six points





The heap *integrates* information about x- and y-coordinates.

- Points towards the top are more to the left.
- Points in the left child are in the lower half.
- Points in the right child are in the upper half.

Priority Search Trees

1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

definition of a priority search tree

Let P be a set of n points in the plane.

1. $p_{\min} \in P$ has the smallest x-coordinate.
2. y_{mid} is the median of the y-coordinates of $P \setminus \{p_{\text{mid}}\}$.
3. With $p_{\min}$ and y_{mid}, define
   - $P_{\text{below}} = \{ p \in P \setminus \{p_{\min}\} \mid p_y < y_{\text{mid}} \}$, and
   - $P_{\text{above}} = \{ p \in P \setminus \{p_{\min}\} \mid p_y > y_{\text{mid}} \}$.

Definition (priority search tree)

The *priority search tree T for P* is defined recursively as

1. if $P = \emptyset$, then T is an empty leaf, otherwise
2. the root v of T stores $(p_{\min}, y_{\text{mid}})$
   - the left child of v is a priority search tree for P_{below}, and
   - the right child of v is a priority search tree for P_{above}.

making a priority search tree

Algorithm MAKEPRIORITYSEARCHTREE(P)

Input: $P = \{p_1, p_2, \ldots, p_n\}$, a set of n points in the plane.
Output: the root of the priority search tree for P.

if $P = \emptyset$ then return empty leaf
else
  let $p_{\rm min}$ be the leftmost point of P: $p_{\rm min} = p \in P$, $p_x = \min\limits_{q \in P} q_x$
  $y_{\rm mid}$ is the median of the y-coordinates of $P \setminus \{p_{\rm mid}\}$
  $v = \text{NODE}(p_{\rm min}, y_{\rm mid})$
  $P_{\rm below} = \{ \, p \in P \setminus \{p_{\rm min}\} \mid p_y < y_{\rm mid} \, \}$
  $P_{\rm above} = \{ \, p \in P \setminus \{p_{\rm min}\} \mid p_y > y_{\rm mid} \, \}$
  LEFTCHILD(v) = MAKEPRIORITYSEARCHTREE($P_{\rm below}$)
  RIGHTCHILD(v) = MAKEPRIORITYSEARCHTREE($P_{\rm above}$)
  return v

the cost of making a priority search tree

Lemma (cost of making a priority search tree)

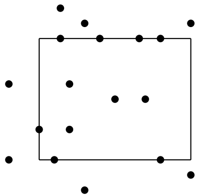
For a set P of n points,
algorithm MAKEPRIORITYSEARCHTREE *has running time $O(n \log(n))$.*
If the points in P are sorted on their y-coordinate,
then the priority search tree can be constructed in $O(n)$ time.

Exercise 1: Consider the set of 15 random points: $P = \{ (46, 32),$ $(63, 73), (87, 66), (83, 92), (44, 41), (64, 74), (46, 35), (45, 24),$ $(27, 43), (52, 54), (90, 84), (78, 84), (72, 28), (38, 65), (61, 57) \}$.

- Construct the priority search tree for P.
- Sort the points in P on their y-coordinate.
  Does the construction of the priority tree go faster after sorting?
- Formulate the algorithm to construct a priority search tree for a list P, of points sorted on their y-coordinate.

points with the same x- or y-coordinates

Taking the leftmost point and splitting the set of other points on the median y-coordinate leads to a tree.
However, our points are end points of line segments.



What if points

- *have the same x-coordinate?*
- *have the same y-coordinate?*

Exercise 2:
Does the technique of composite coordinates solve the problem?
Consider eight points on the boundary of a square:

$$(0,0), (1,0), (2,0), (0,1), (2,1), (0,2), (1,2), (2,2).$$

Define a heap for this set of points.

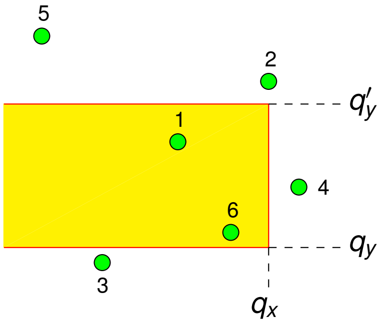
Priority Search Trees

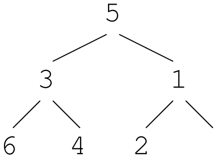
1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

a query window

Input: $P = \{ p_1, p_2, \ldots, p_n \}$, a set of n points in the plane; and
a query window $W = (-\infty : q_x] \times [q_y : q'_y]$.
Output: $P \cap W$.





reporting in subtrees

Given the query window $W = (-\infty : q_x] \times [q_y : q'_y]$, we report the points $p = (p_x, p_y)$ with $p_y \in [q_y : q'_y]$ if they pass the test $p_x \leq q_x$.

Algorithm ReportInSubTree(v, q_x)

Input: root v of a priority search tree,
q_x is the right bound on x of a query window.
Output: all points p with $p_x \leq q_x$.

1. if not IsLeaf(v) and $p(v)_x \leq q_x$ then
2. Report $p(v)$
3. ReportInSubTree(LeftChild(v), q_x)
4. ReportInSubTree(RightChild(v), q_x)

the cost of reporting in subtrees

Lemma (cost of reporting in subtrees)

Algorithm ReportInSubTree(v, q_x) *takes* $O(1 + k_v)$ *time to report* k_v *points* p *with* $p_x \leq q_x$.

- By the definition of the heap,
  nodes closer to the root are more to the left.
  Therefore, as soon as $p(v)_x > q_x$,
  the children of v are also to the right of q_x and need not be visited.
- At any node v, we spend $O(1)$ time:
  1. We test: if not IsLeaf(v) and $p(v)_x \leq q_x$.
  2. If the test passes, then report and visit the children.
- By the test $p(v)_x \leq q_x$ all reported points lie to the left of q_x.
  Every reported point has at most two children. We visit at most twice as many nodes as the number of reported ones.

a split vertex

At each node we store

- $p_{\rm min}$ the leftmost point, and
- $y_{\rm mid}$ the median of the y-coordinates of the remaining points.

The points below $y_{\rm mid}$ are in the left child,
The points above $y_{\rm mid}$ are in the right child.

With the query window $W = (-\infty : q_x] \times [q_y : q'_y]$,

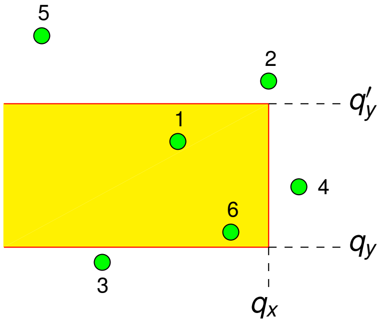
- if $y_{\rm mid} < q_y$, then we do not visit the left child,
- if $y_{\rm mid} > q'_y$, then we do not visit the right child.

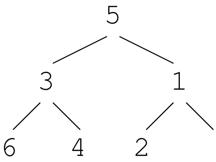
As long as $y_{\rm mid} < q_y$ or $y_{\rm mid} > q'_y$,
the search path is one single branch of the priority search tree.

The node where both children must be visited is a *split vertex*.

an example of a split vertex

Consider our running example with the query window in yellow:





What is the split vertex?

query a priority search tree

Algorithm QUERYPRIORITYSEARCHTREE(T, W)

Input: T, a priority search tree,
$W = (-\infty : q_x] \times [q_y : q'_y]$, a query window.
Output: all points of $T \cap W$.

1. let $v_{\rm split}$ be the split vertex
2. for each node v on the search path from q_y to q'_y do
3.   if $p(v) \in W$, then REPORT $p(v)$
4. for each node v on path of q_y in LEFTCHILD($v_{\rm split}$) do
5.   if path goes left at v then
       REPORTINSUBTREE(RIGHTCHILD(v), q_x)
6. for each node v on path of q'_y in RIGHTCHILD($v_{\rm split}$) do
7.   if path goes right at v then
       REPORTINSUBTREE(LEFTCHILD(v), q_x)

illustrate the query algorithm

Exercise 3: Consider the set of 15 random points: $P = \{ (46, 32)$, $(63, 73)$, $(87, 66)$, $(83, 92)$, $(44, 41)$, $(64, 74)$, $(46, 35)$, $(45, 24)$, $(27, 43)$, $(52, 54)$, $(90, 84)$, $(78, 84)$, $(72, 28)$, $(38, 65)$, $(61, 57) \}$.

Exercise 1 asks to construct a priority search tree for P.

1. Illustrate algorithm QUERYPRIORITYSEARCHTREE with a well chosen example of a query window.
2. Run algorithm QUERYPRIORITYSEARCHTREE step by step on the priority search tree for P and your chosen query window.

   In running the algorithm, identify $v_{\rm split}$ and refer to the steps in the algorithm.

the cost to query a priority search tree

Lemma (cost to query a priority search tree)

The algorithm QUERYPRIORITYSEARCHTREE *on a search tree for n points reports k points in the query window in $O(\log(n) + k)$ time.*

- All reported points lie in the query window.
- The algorithm will report *all* points, none are missed.
- The algorithm is output sensitive.
- The depth of the heap is $O(\log(n))$,
  because of the split on the median.

the cost of priority search trees

We summarize the results in the following.

Theorem (cost of priority search trees)

A priority search tree for n points

- *uses $O(n)$ storage, and*
- *takes $O(n \log(n))$ time to construct.*

The query time is $O(\log(n) + k)$,
where k is the number of reported points.

Priority Search Trees

1. Storing Points in the Plane
   - data structures for windowing queries
   - windowing queries using a heap

2. Priority Search Trees
   - definition and construction
   - query a priority search tree
   - running an example of CGAL

running an example of CGAL

On a Window Subsystem for Linux (WSL), running Ubuntu:

1. Run `sudo apt-get install` followed by `libgmp-dev`, `libmpfr-dev`, `libcgal-dev`.
2. Download `CGAL-5.6-examples.zip`.
3. Compile `nearest_neighbor_searching.cpp` from the `Spatial_searching` folder.

This example constructs a tree and then queries the tree for the nearest point.

Exercise 4: Construct trees for sufficiently large point sets.
Report the observed times for the construction of the data structure.
Do you observe the $O(n \log(n))$ as n grows?

summary and exercises

Storing n points in a heap has storage cost $O(n)$, construction cost $O(n \log(n))$ and query time $O(\log(n) + k)$, where k is the size of the output.

We covered section 10.2 in the textbook.

Consider the following activities, listed below.

1. Write the solutions to exercise 1, 2, 3, and 4.
2. Read the CGAL documentation on the dD Spatial Searching.
3. Consider the exercises 10.2, 10.3, 10.10 in the textbook.